Jambon F. La prévention d'erreur par la technique des "Fonctions de Contrainte". ***Onzième conférence sur l'Interaction Homme-Machine (IHM'99), Montpellier, France***, 22-26 novembre 1999. p. 102-109.

La prévention d'erreur par la technique des «Fonctions de Contrainte»

Francis JAMBON

LISI/ENSMA
Téléport 2, 1 av. Clément Ader, B.P. 40109
86961 Futuroscope Chasseneuil Cedex, France
Francis.Jambon@ensma.fr
Téléphone : (33/0) 5 49 49 80 70
Télécopie : (33/0) 5 49 49 80 64

RÉSUMÉ

Nous présentons, en suivant une approche technique, une méthode de prévention d'erreur : les fonctions de contraintes. Ces fonctions visent à bloquer les actions de l'utilisateur en situation d'erreur. Dans un premier temps, nous faisons un tour d'horizon de la bibliographie sur le sujet, puis nous proposons une classification des différents types de fonctions de contrainte que nous illustrons d'exemples aussi bien tirés des systèmes à risques, que des logiciels grand public. Nous proposons ensuite un guide d'usage des fonctions de contrainte en ingénierie des IHM dirigée par les besoins des concepteurs. Nous terminons l'article par une discussion sur la nécessité d'une approche ergonomique en amont de l'usage des fonctions de contrainte. Nous mettons en particulier l'accent sur les besoins d'information et d'explication lors de la détection d'erreurs humaines.

MOTS CLÉS : Fonction de contrainte, détection d'erreur, prévention d'erreur, classification, guide d'usage, ingénierie des IHM.

INTRODUCTION

L'erreur humaine, souvent qualifiée de "faute", est invoquée régulièrement comme cause principale d'accidents aux conséquences dramatiques dans les systèmes à risques (transports, chimie, énergie, etc.). La "lettre ouverte à un futur ami" de J.-L. Nicolet, A. Carnino et J.-C. Wanner [9] illustre cet état de fait. Ainsi, la responsabilité d'une catastrophe est attribuée presque exclusivement à l'opérateur final : l'humain qui a commis la dernière action ayant déclenché l'accident est blâmé. Cette attitude revient à attribuer la responsabilité d'une diffamation dans un article de presse aux marchands de journaux, qui certes ont contribué à sa diffusion, mais n'en sont pas les principaux responsables.

Cet état d'esprit tend à se raréfier dans le cadre des systèmes à risques. En effet, on reconnaît aujourd'hui l'existence d'erreurs latentes au sein des organisations au sens de J. Reason [14] (Chap. 7). Parmi ces erreurs latentes, la mauvaise conception de l'interface homme-machine du système, ayant conduit à favoriser l'erreur de l'opérateur final, est ainsi mise en lumière. Cependant, si le coût humain ou dans le meilleur des cas financier, d'une catastrophe incite les industries à risques à s'intéresser aux "facteurs humains" lors de la conception des systèmes, cette démarche est difficilement admise par l'industrie du logiciel non critique, qui fait généralement porter le coût de l'erreur sur l'utilisateur final. Par exemple, le fichier d'une lettre perdu par une secrétaire brise, au pire, le calme d'un bureau. Pourtant, le coût résultant du temps perdu par celle-ci est souvent du même ordre de grandeur que le prix du logiciel incriminé !

Face à l'erreur humaine, lorsque les opérationnels se rendent compte que la "Blame and Train Philosophy" [11] ou l'automatisation systématique [3] n'apportent en fait que des progrès mitigés, une approche prenant en compte les facteurs humains devient alors la seule issue. L'offre actuelle en termes de facteurs humains peut, selon R. Amalberti [1], se positionner en trois grand types d'approches :

- *L'approche médicale*, dans la lignée de la médecine du travail, propose une vision très ciblée des problèmes, par exemple la prévention des Troubles Musculo-Squelettiques. Cette approche est indispensable, mais difficile à étendre aux aspects plus pluridisciplinaires des facteurs humains.
- *L'approche ergonomique*, quant à elle, fait figure de référence. Par la richesse de ses multiples pratiques, elle tend à adapter l'outil à la variabilité humaine. Cette dernière est acceptée comme une richesse, et non considérée comme un problème que l'on doit canaliser à tout prix.
- *L'approche technique*, enfin, se donne pour finalité l'augmentation des performances du couple homme-machine. Cette approche technique du facteur humain développe des solutions techniques pragmatiques aux problèmes de facteurs humains sans pour autant en épouser (et même parfois en comprendre) les fondements. Malgré les nombreuses critiques que l'on peut lui faire, en particulier son manque d'humanité, elle a su se montrer d'une réelle efficacité.

Malgré ces critiques, c'est de cette dernière approche dont ce réclame cet article. En effet, et en connaissance de cause, il propose une solution purement technique à une problème de facteur humain : la détection d'erreur par l'opérateur. Cependant, bien loin de négliger les apports de l'approche médicale ou ergonomique, les solutions techniques proposées ici doivent être considérées de la même manière que les boîtes à outils en programmation des IHM, ou les briques en construction de bâtiments : un élément indispensable mais qu'il faut savoir utiliser pour obtenir un résultat cohérent. Un programmeur n'a pas forcément de compétences en ergonomie, tout comme un maçon en architecture.

Cet article se propose de dresser le portrait d'une technique de prévention des erreurs, les fonctions de contraintes, et d'en proposer une taxonomie et un guide d'usage. Dans un premier temps, à partir d'une étude bibliographique, les différentes méthodes utilisées par un opérateur dans la détection des erreurs sont passées en revue à partir d'approches orientées ergonomie. Puis, les connaissances actuelles sur la technique des fonctions de contraintes sont détaillée en prenant le point de vue à la fois de ergonomie, mais aussi de ingénierie. Dans un second temps, et selon une approche purement technique, une classification des fonctions de contrainte est proposée ainsi qu'un guide d'usage. Enfin, une réflexion sur les liens entre ergonomie et ingénierie est proposée sur l'usage de cette technique de prévention d'erreur.

LA DÉTECTION D'ERREUR

L'erreur est humaine, et malgré tous nos efforts, les erreurs des opérateurs sont inévitables. Au-delà de ce constat négatif, nous posons que les erreurs sont aussi tout à fait acceptables. En effet, de nombreux auteurs considèrent que les erreurs sont le prix à payer pour l'extraordinaire habileté et capacité d'adaptation dont peut faire preuve l'être humain, même dans les conditions les plus difficiles, là où la machine est prisonnière de sa logique limitée.

L'occurrence des erreurs acceptée, la responsabilité du concepteur d'un système à risque consiste à en minimiser les conséquences potentielles. Selon D. Zapf et J. Reason [18], le traitement des erreurs peut se diviser en deux étapes essentielles (Cf. fig. 1) : le diagnostic de l'erreur, puis la correction de celle-ci. Dans cet article, nous nous intéressons principalement au diagnostic, et plus précisément à la détection. Dans un premier temps nous décrivons quels sont les différents modes de détection d'erreur, puis nous détaillons comment l'environnement peut aider cette détection. Enfin, nous nous intéresserons plus particulièrement aux fonctions de contrainte.

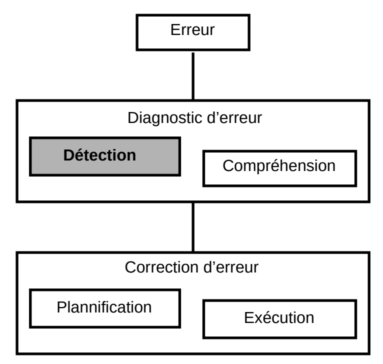


Figure 1 : Traitement d'erreur (Fig. traduite de [18]).

Modes de détection d'erreur

Selon J. Reason [14][1] (Chap. 6), l'erreur d'un opérateur peut être détectée selon trois principaux modes :

- *L'autocontrôle* permet à l'opérateur d'effectuer un contrôle réactif sur l'état du système. C'est un comportement dirigé par l'erreur : l'écart détecté entre l'état souhaité et l'état observé est évalué par l'opérateur qui effectue en retour les actions nécessaires. Ce type de processus est assez mal connu mais semble être utilisé dans le contrôle de la posture, les erreurs de la parole, la détection des actions non désirées (ratés de l'action) ou encore la résolution de problème. C'est avant un mode de détection interne à l'opérateur.
- *La détection par des tiers* est au contraire, le fait opérateurs externes à l'action initiale. Ce mode de détection est très fréquent dans le cas d'erreur de diagnostic. Dans une telle situation, ce sont des "esprits neufs", non soumis aux mêmes "fixations" que les opérateurs fautifs, qui détectent l'existence de données contredisant le diagnostic initial.
- *Les indications de l'environnement* permettent à l'opérateur de détecter une erreur, ou au moins de se rendre compte d'une anomalie grâce aux comportements perceptibles du système. Ces comportements peuvent être naturels au fonctionnement du système : un réfrigérateur qui ne fonctionne pas n'est "naturellement" pas froid. Ils peuvent aussi avoir été créés à dessein par le concepteur : un automobiliste qui oublie ses phares allumés, se fait habituellement rappeler à l'ordre lors de l'ouverture de sa portière grâce à un signal sonore.

Les indications de l'environnement sont le mode de détection d'erreur qui nous intéresse dans notre approche "technique" des facteurs humain. En effet, ce mode est le seul à pouvoir être introduit volontairement au cours du processus de conception d'un système. Il a donc un rôle primordial en ingénierie des IHM destinées aux systèmes critiques. Selon J. Reason, les indications de

[1] L'ouvrage référencé dans cet article est la traduction française effectuée par J.-M. Hoc de la version originale «Human Error» [13].

l'environnement peuvent se voir selon trois axes : les indices, la réponse des systèmes, et les fonctions de contrainte.

La détection à partir d'indices

Lors d'une activité de résolution de problème, J. Reason signale que l'environnement peut permettre à l'opérateur de découvrir non pas une réelle erreur, mais plutôt un plan d'actions sous-optimal. Il cite en particulier une expérience personnelle où il a dû retirer le cric de sous sa voiture pour s'apercevoir, une fois celle-ci au sol, que le déblocage des écrous de ses roues était plus aisé. Selon l'auteur, ce mode de détection n'est pas véritablement une fonction de contrainte, car l'environnement lui a simplement donné une occasion de rejoindre un plan d'action plus optimal. Nous verrons que nous ne partageons pas tout à fait ce point de vue.

La réponse des systèmes

C. Lewis et D. Normal [7] ont défini six types de réponses possibles d'un système lors d'une détection d'erreur :

- *Les baillons («GAG»)*[2] sont reconnus comme des fonctions de contrainte. Leur rôle est de bloquer tout comportement qui s'écarte des réponses attendues. Par exemple, l'insertion incorrecte (à l'envers) d'une disquette 3"1/2 est impossible dans un lecteur classique, car celle-ci est bloquée en cours de route par un dispositif mécanique.
- *Les avertissements («Warn»)* préviennent l'utilisateur sans pour autant le bloquer. Ce type de réponse peut être vu comme un baillon modéré. En effet, l'utilisateur est libre de passer outre l'avertissement. Un exemple d'avertissement bien connu est l'utilisation du signal sonore d'oubli des phares. En informatique, ces avertissements sont utilisés couramment afin de protéger les actions jugées potentiellement dangereuses, comme vider la corbeille.
- *L'inertie («Do nothing»)* est une réponse "silencieuse" aux erreurs de l'utilisateur : le système ne fait rien. Cette technique est utilisée dans le cadre de la manipulation directe, où, lorsqu'une action sur un objet est invalide, elle ne se fait tout simplement pas. Très aisée à implémenter, l'inertie a un défaut majeur : elle laisse le soin à l'utilisateur de découvrir ce qui ne va pas.
- *L'autocorrection («Self correct»)* est au contraire une réponse active, et même parfois intrusive du système : celui-ci prend l'initiative de modifier les entrées de l'utilisateur dans le "bon" sens. Le système doit obligatoirement disposer d'une fonction "défaire" afin de permettre à l'utilisateur de remédier aux éventuelles corrections non-valides. Ce type de réponse est utilisé dans certains éditeurs, comme par exemple Microsoft™ Word98, pour corriger les fautes de frappe courantes : "ofre" devient "offre", ou "oeuf" devient "œuf", etc.
- *Le dialogue («Let's talk about it»)* initié entre le système et l'utilisateur doit permettre à ce dernier de choisir la réponse à apporter au problème détecté. Ce type de réponse est utilisable lorsqu'il existe un nombre relativement faible de possibilités. Par exemple, lors de l'ouverture d'un fichier de type inconnu, le "Finder" du Macintosh propose une liste d'applications possibles.
- *L'enseignement («Teach me»)* est un type de réponse plus complexe à implémenter, il est utilisé dans des systèmes d'interrogation en langue naturelle : le système demande à l'utilisateur de lui définir tout mot absent de son vocabulaire, jusqu'à ce que l'ensemble des mots utilisés soient connus.

[2] Les traductions ont été empruntées à [14].

Les fonctions de contrainte

La notion de *fonction de contrainte* semble avoir été introduite par C. Lewis et D. Norman dans l'article «Designing for Error» [7]. Cet article bien connu définit la "propriété" de fonction de contrainte comme «ce qui bloque la poursuite du comportement tant que le problème n'a pas été corrigé»[3]. Dans cet article, C. Lewis et D. Norman se contentent de cette définition et l'illustrent par des exemples. Parmi ces exemples, ils citent les *baillons* définis précédemment. De cette définition à très large spectre, on peut relever deux points intéressants. En premier lieu, la notion de blocage est essentielle, il d'agit ici d'interrompre brutalement le fil des actions de l'utilisateur en cas d'erreur, ou plus généralement de "problème". En second lieu, la correction apparaît comme une étape obligatoire, car le fil des actions ne peut continuer qu'à l'issue ce cette correction.

D. Norman, quelques années plus tard [10], a proposé une autre définition quelque peu plus restrictive des fonctions de contrainte. Une fonction de contrainte est ainsi définie : «Les fonctions de contrainte sont une sorte de contrainte physique : ce sont des situations dans lesquelles les actions subissent des contraintes de telle manière qu'un échec à une étape empêche l'étape suivante de se produire»[4]. Cette définition est plus restrictive au sens où l'obligation de correction a disparu, et la notion de comportement a été réduite à la notion de tâche. Cependant, nous retrouvons l'idée essentielle du blocage.

[3] Traduction proposée dans [14] p. 222 de «…something that prevent the behavior from continuing until the problem has been corrected» in [7] p. 420.

[4] «Forcing function are a form of physical constraint: situations in which the actions are constrained so that a failure at one stage prevent the step from happening».

A ce niveau il est possible de voir dans la fonction de contrainte une généralisation de la technique des détrompeurs, lesquels sont très utilisés en mécanique pour la prévention des erreurs de montage. En corollaire, remarquons que le domaine de la prévention des erreurs en sécurité industrielle utilise les fonctions de contrainte depuis semble-t-il des dizaines d'années. Les dispositifs utilisés sont très simples, et se basent sur des systèmes mécaniques, électromécaniques et parfois maintenant électroniques. En outre D. Norman, en s'inspirant de ces méthodes utilisées en sécurité industrielle, décrit trois types de fonctions de contrainte : les interblocages, les verrouillages en fonction et hors fonction. Nous les détaillons ci-dessous en les illustrant d'exemples tirés du domaine informatique.

Les interblocages («interlock») permettent d'imposer une séquence entre deux actions de l'utilisateur. En sécurité industrielle, les interblocages permettent par exemple d'obliger un opérateur à couper l'alimentation électrique d'un appareil avant de l'ouvrir. Certains Macintoshes sont fabriqués de cette façon : il n'est pas possible d'ouvrir le boîtier sans avoir enlevé le cordon d'alimentation secteur. Mais ce type de fonction de contrainte se retrouve également dans nos interfaces homme-machine : l'inactivation d'une commande, visible par le grisé d'un item de menu, est une fonction de contrainte qui empêche l'utilisateur d'activer la commande, par exemple tant qu'un objet valide n'a pas été préalablement sélectionné.

Les verrouillages en fonction («lockin») maintiennent un système actif malgré la demande de l'utilisateur.

Citons ici un exemple intéressant : il s'agit d'un système de navigation par satellite destiné à l'aéronautique. Le système en question met quelques minutes pour se remettre en route. C'est pourquoi, afin d'éviter une fausse manœuvre entraînant une extinction prématurée, les concepteurs ont ajouté un délai lors de l'extinction : le pilote doit appuyer plus de trois secondes sur le bouton d'extinction pour arrêter le système. Au cours de cette procédure d'extinction l'écran affiche un décompte afin d'en informer le pilote. Si le pilote relâche le bouton avant la fin du décompte, le système reprend son fonctionnement normal, dans l'état exact où il était auparavant [2]. Ce système de maintient obligatoire a été probablement préféré à une simple confirmation, car une double erreur de doigt est toujours possible.

Plus proche de l'informatique, l'ancestral éditeur sous UNIX "vi" empêche l'utilisateur de quitter le logiciel si le fichier en cours d'édition a été modifié et non sauvé. De plus, il informe l'utilisateur du problème rencontré par le message : «Pas de sauvegarde effectuée depuis la dernière modification (l'ordre :quit! passe outre).»[5].

Les verrouillages hors fonction («lockout») empêchent une action isolée d'être effectuée. Dans le domaine de la sécurité domestique, on peut citer les portes de machines à laver qui refusent de s'ouvrir tant que le tambour est en rotation. Le logiciel de lecture de courrier électronique Eudora™ en est un autre exemple : lorsque l'utilisateur s'échine à taper sur son clavier alors qu'aucune fenêtre n'est à même d'interpréter les caractères, le logiciel les ignore et prévient l'utilisateur par un son.

Synthèse et critique

En résumé, la détection d'une erreur humaine peut s'effectuer selon trois modes : par autocontrôle, par l'intervention de personnes tierces, ou encore grâce aux indications de l'environnement. Ces indications peuvent être vues comme des fonctions de contrainte, des indices, ou être une réponse du système. Ces trois dimensions ne sont pas orthogonales. Par exemple, les *baillons* sont une des réponses possibles d'un système, mais sont également une fonction de contrainte. Lorsque l'on détaille les fonctions de contrainte, les *verrouillages en fonction* sont assez proches de la notion *d'inertie*. De plus, un *verrouillage* empêchant de quitter une application tant qu'un fichier n'est pas sauvegardé peut être aussi vu comme un *interblocage* entre les actions sauvegarder et quitter. C'est pourquoi dans un premier temps, nous avons cherché à clarifier la notion de fonction de contrainte en proposant une classification.

Lorsque, du point de vue d'un concepteur, l'on s'intéresse à la réponse que peut avoir un système aux erreurs de l'utilisateur, ces classifications et exemples ne sont pas toujours directement utilisables, car ils ne s'intéressent pas au fond du problème en ingénierie, c'est à dire le besoin ponctuel du concepteur. A. Rizzo, D. Ferrante, et S. Bagnara ont proposé une approche intéressante du traitement d'erreur, ils ont ainsi défini sept recommandations [15]. Cependant, ces recommandations sont d'ordre général et n'offrent pas de solution prête à l'emploi. Au contraire, les notions d'interblocage et de verrouillage proposés par D. Norman apportent des exemples de solutions. Néanmoins, elles ne permettent pas de savoir quelles raisons ont poussé à les utiliser. C'est face à ce double constat que, dans un second temps, nous avons cherché à définir un guide d'usage des fonctions de contrainte à l'usage de l'ingénierie des IHM.

CLASSIFICATION

Notre classification se base sur quatre axes que nous supposons orthogonaux : la force du blocage, sa relation temporelle avec l'action exécutée par l'utilisateur, l'effet recherché, et enfin son type.

[5] «No write since last change (:quit! overrides)».

Force du blocage

La force du blocage exprime la capacité de contrainte exercée par le dispositif :

- *Total :* l'action ne peut pas s'effectuer. Par exemple, il est impossible d'ouvrir la porte d'une machine à laver tant que le tambour est en rotation. De même, il est impossible d'activer une commande si l'item de menu lui correspondant a été grisé.
- *Total transgressable :* l'action peut s'effectuer, mais l'utilisateur doit volontairement en prendre l'initiative de transgresser une action jugé interdite, sauf cas exceptionnel, par le concepteur. Par exemple, il est quasiment impossible d'ouvrir la porte d'un train si celui-ci est en marche, sauf en utilisant l'ouverture d'urgence, laquelle est plombée. De même, il est toujours possible de retirer la disquette d'un Macintosh avec un trombone déplié.
- *Entrave :* l'action est possible, mais une difficulté supplémentaire a été volontairement ajoutée. Ainsi, les bouteilles de produits dangereux peuvent être ouvertes grâce à une méthode complexe qui n'est pas à la portée des enfants. La Corbeille du Macintosh, si elle contient des fichiers verrouillés, peut être vidée de force en appuyant simultanément sur la touche "option".
- *Informatif :* l'action est possible, mais l'utilisateur est informé qu'il prend un risque, le système lui laisse alors le choix de revenir sur sa décision. Le bouton d'arrêt de caméras vidéo professionnelles ou de photocopieurs est souvent protégé par un clapet qu'il faut soulever pour pouvoir accéder à l'interrupteur. De même, le "Finder" du Macintosh demande systématiquement confirmation avant l'arrêt de la machine.

Relation temporelle

La relation temporelle situe l'action de la contrainte par rapport à celle de l'utilisateur :

- *Avant :* la fonction de contrainte s'exprime avant que l'utilisateur n'ait pu agir sur le système. Par exemple, les détrompeurs utilisés en mécanique empêchent le montage incorrect d'une pièce avant que celui-ci ne soit effectivement réalisé. De même, les items de menus grisés ne permettent pas à l'utilisateur d'activer la commande.
- *Après :* la fonction de contrainte ne prend effet à l'issue de l'action de l'utilisateur. Cette relation temporelle semble difficilement réalisable en mécanique, par contre, elle est très utilisée en informatique. Par exemple, les systèmes d'exploitation demandent généralement confirmation à l'utilisateur avant l'initialisation d'une disquette.
- *Pendant :* la fonction de contrainte agit en même temps que l'utilisateur. Sur certains avions, les robinets d'arrivés d'essence et d'huile sont montés de telle façon que l'ouverture de l'essence entraîne obligatoirement, par le jeu d'un ergot, l'ouverture de l'huile en même temps. De même, certaines cases à cocher liées sont activées automatiquement par le système en même temps que les actions de l'utilisateur.

Effet recherché

L'effet recherché détermine le type de contrainte sur la ou les actions de l'utilisateur :

- *Empêcher :* la fonction a pour but de garantir la non exécution d'une action de l'utilisateur. Par exemple, les extincteurs disposent d'une goupille de sécurité empêchant leur utilisation involontaire. De même, il n'est pas possible sur un Macintosh de mettre le disque contenant le système à la Corbeille.
- *Forcer :* la fonction a pour but d'obliger l'utilisateur à exécuter une action déterminée. Ce type de contrainte est utilisé pour les portes de secours où il n'est possible de pousser la porte qu'en appuyant sur une barre, laquelle déclenche le penne de la porte. Ce type d'obligation se rencontre également dans le cas des boîtes de dialogue modales où un seul bouton est disponible.

Type de blocage

Le type de blocage détermine si celui-ci met en jeu une ou plusieurs actions de l'utilisateur sur le système :

- *Verouillage:* la fonction a pour but de contraindre l'utilisateur à effectuer (*verrouillage en fonction*) ou à ne pas effectuer (*verrouillage hors fonction*) une action isolée sur le système.
- *Interblocage:* la fonction a pour but de forcer l'utilisateur à effectuer des actions modifiant l'état du système selon une séquence imposée.

Notons que cette distinction est parfois difficile. En effet, bloquer par un verrouillage en fonction un traitement de textes tant qu'aucun fichier n'est ouvert, revient à réaliser un interblocage entre les actions quitter et fermer (ou sauvegarder).

USAGE EN INGÉNIERIE DES IHM

Avant de présenter notre guide de conception, et afin d'illustrer notre propos, nous allons présenter la rétro-conception d'un exemple tiré de l'aéronautique.

Rétro-conception d'un exemple

De petits avions sont utilisés couramment dans les aéro-club par des pilotes occasionnels. Ce type d'avion, du point de vue de sa motorisation, se comporte un peu comme une voiture ancienne : un bouton poussoir commande le démarreur et l'allumage est protégé par une clé de contact. En outre, l'arrivée d'essence peut être coupée par un robinet en cas de feu moteur. L'usage veut qu'après chaque vol l'arrivée d'essence soit coupée afin d'éviter tout risque d'incendie. Cependant, malgré l'usage des check-lists, il s'est avéré qu'un nombre significatif de

pilotes oubliaient d'ouvrir le robinet lors du redémarrage de l'avion. Sur de petits terrains d'aviation, il reste assez de carburant dans le carburateur et les tuyauteries pour permettre le roulage et un début de décollage. L'avion peut ainsi se retrouver en vol à très basse altitude, sans moteur…

Une première réaction à ce problème fut la diffusion de la consigne de ne jamais couper l'essence, les risques d'incendie à l'arrêt ayant moins de conséquences dramatiques qu'un arrêt moteur en vol. Quelques années plus tard, un constructeur a introduit un nouveau type de robinet d'essence, lequel intègre le bouton du démarreur masqué par la position fermée du robinet (Cf. fig. 2). Le constructeur a voulu empêcher les pilotes de décoller avec l'essence fermée. Pour ce faire, il a fallu lister toutes les tâches absolument nécessaires au décollage d'un avion qui ne peuvent être oubliées : mettre le contact et appuyer sur le bouton du démarreur. Ainsi, le bouton du démarreur fut choisi comme fonction de contrainte obligeant le pilote à ouvrir l'arrivée d'essence avant de faire décoller l'avion.



Figure 2 : Robinets d'arrivée d'essence ancien modèle en haut et nouveau modèle en bas ; ce nouvreau modèle possède une fonction de contrainte qui oblige le pilote à ouvrir l'arrivée d'essence pour pouvoir actionner le démarreur (photographie en bas à droite).

Démarche de conception

L'exemple précédent dresse les grandes lignes de ce que peut être la démarche d'un concepteur face à une d'erreur possible de l'utilisateur. En effet, le concepteur doit tout d'abord **identifier** sans ambiguïté la tâche potentiellement génératrice d'erreur, dans notre exemple c'est l'oubli de l'ouverture de l'essence qui est incriminée.

Il doit ensuite définir **l'effet recherché** sur cette tâche, i.e. s'il faut forcer ou empêcher celle-ci. Dans notre exemple, il s'agit de forcer l'ouverture de l'essence. Notons que cet effet recherché est lié au choix initial de la tâche génératrice d'erreur. En effet, forcer l'utilisateur à sauvegarder un fichier avant de quitter est équivalent à empêcher celui-ci de quitter sans avoir sauvegardé.

Le contexte détermine ensuite **le type** de la contrainte. Deux cas se présentent :

- *Le contexte est dépendant du système seul ou de l'utilisateur seul.* Par exemple, la rotation du tambour d'une machine à laver est seulement dépendante du programmateur de cette machine. Dans ce cas, le concepteur doit avoir recours à un *verrouillage*.
- *Le contexte est dépendant d'une action de l'utilisateur sur le système.* Dans ce cas, le concepteur doit avoir recours à un *interblocage*. S'il s'agit d'empêcher une action, il lui faut déterminer dans quel contexte l'action devient valide, et forcer (ou inciter) l'utilisateur à modifier le contexte avant. En corollaire, s'il s'agit de forcer une action, il lui faut trouver dans quel contexte l'action devient invalide, et obliger l'utilisateur à exécuter son action avant le changement de contexte.

Puis, **la force** du blocage doit être ajustée au niveau de protection requis, c'est-à-dire le plus souvent aux conséquences potentielles de l'erreur. Cette force doit aussi prendre en compte les cas exceptionnels : un blocage total n'est pas toujours justifié, et peut même parfois s'avérer dangereux. En outre, la force doit être choisie avec soin en fonction de l'activité réelle de l'opérateur : une confirmation systématique d'une action régulièrement effectué n'est plus lue. La force de la contrainte est probablement la dimension la plus difficile à déterminer, et doit s'effectuer en collaboration avec des ergonomes.

Enfin, **la position** de la contrainte est le plus souvent déterminée par les capacités techniques du système. Les dispositifs mécaniques facilitent une fonction de contrainte avant l'action, souvent par le jeu d'ergots. Au contraire, les dispositifs informatiques rendent plus difficile cette position : il est en effet difficile de bloquer une souris ou un clavier sans dispositifs électromécaniques complexes. L'action incriminée est donc souvent bloquée après avoir été effectuée par l'utilisateur. Cependant, les items de menus grisés, ou les boutons inactifs sont des fonctions de contrainte avant l'action très facilement implémentables. Parfois, l'information nécessaire n'est pas toujours disponible, et le système doit, au moment de l'action, se renseigner sur la validité de celle-ci. On obtient alors une contrainte postérieure à l'activation de l'action.

Information & Explication

Les fonctions de contrainte ont généralement un défaut majeur : l'utilisateur est bloqué, mais parfois n'en est pas informé, et le plus souvent ignore quelle est la raison du blocage. Le principe fondamental d'une fonction de contrainte est certes le blocage, mais il n'est pas rare de voir des utilisateurs utiliser la force brute afin de passer outre une fonction de contrainte. Ainsi, les lecteurs de disquettes sont régulièrement forcés par les débutants cherchant à tout prix à mettre la disquette dans le mauvais sens.

Le logiciel Eudora™ par exemple inclut les deux informations de manière astucieuse. Si un utilisateur frappe une touche du clavier, le son d'alarme standard est émis à chaque frappe clavier. Cependant, le rôle de ce son est limité, en effet, il informe uniquement que l'action de l'utilisateur n'a pas été prise en compte, pas la cause de cette erreur. Ainsi, les concepteurs ont ajouté une fenêtre d'explication (Cf. fig. 3) qui ne se déclenche que lorsque le temps entre deux frappes est inférieur à un certain seuil, synonyme que l'utilisateur commence à rédiger une lettre. Ce type de conception révèle une analyse fine de la tâche de l'utilisateur, mais aussi un respect des règles ergonomiques de conception classiques [16].

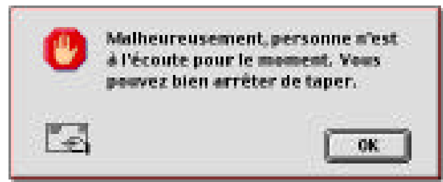


Figure 3 : Message d'alerte de Eudora™ lorsque l'utilisateur entre des caractères non utilisables par le logiciel.

Exemple d'application

Appliquons maintenant la notion de fonction de contrainte à un exemple de la vie courante : il n'est pas rare qu'un utilisateur oublie sa carte à puce dans le lecteur d'un téléphone public à la fin d'une conversation.

Analysons le problème[6]. *L'effet recherché* est ici de forcer l'utilisateur à retirer sa carte. Le contexte est la conversation téléphonique, à l'issu de laquelle il doit retirer la carte. Ainsi, le contexte est dépendant d'une action de l'utilisateur sur le système, le fait de raccrocher. Ceci impose l'interblocage comme *type de contrainte*. La *force* de la contrainte peut être ici totale, le système n'étant pas critique, et de plus il n'existe pas de cas ou il faille raccrocher le combiné avec la carte insérée. Une contrainte après l'action ne pouvant être totale, nous devons implémenter une contrainte avant l'action de raccrocher, c'est-à-dire forcer l'utilisateur à retirer sa carte avant de raccrocher, ou bien empêcher de raccrocher si la carte est encore en place (ce qui revient au même). Il suffit en fait de placer le lecteur de carte à l'emplacement de l'écouteur du combiné téléphonique, empêchant ainsi l'utilisateur de raccrocher le combiné si la carte est encore présente dans son lecteur (Cf. fig. 4).

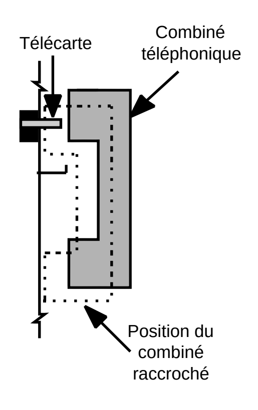


Figure 4 : Application au cas du téléphone public.

Discussion

Face aux erreurs humaines, est apparue une classe d'interfaces dites "tolérant les fautes" [5] p. 212, lesquelles se basent habituellement sur un modèle de tâche embarqué. Ainsi, J. Mo et al. [8] ont proposé un système de prévention des erreurs des utilisateurs qui dispose d'un modèle des plans d'action possible mais aussi d'une liste des tâches activables par l'opérateur en fonction de l'état du système. Ce travail peut se voir comme une généralisation de la notion de fonction de contrainte. Il montre également l'importance lors de la conception de l'analyse de la tâche de l'utilisateur.

L'intégration de dispositifs de protection contre les erreurs au sein d'un système homme-machine à risques peut se voir au premier abord comme la "solution miracle" en termes de sécurité. En effet, l'opérateur est libre d'agir avec le système sans automatisation abusive ni procédures à rallonges, on lui permet même de faire des erreurs, mais on empêche sa bévue d'être catastrophique. Cependant, ce type de protection ne peut concourir à la sécurité que si les opérateurs sont conscients de l'existence du système de protection, et surtout sont capables d'en comprendre la logique de fonctionnement en cours d'action. Ce problème est illustré par les dispositifs de "filets de protection" utilisés en liaison avec les commandes de vol électriques des avions modernes, lesquels maintiennent coûte que coûte l'avion dans son domaine de vol autorisé, et nécessitent une adaptation des procédures d'entraînement des équipages [17].

Enfin, notons que le vocabulaire utilisé entre les ergonomes et les concepteurs n'est pas toujours équivalent. Du point de vue de l'ergonome, une erreur a lieu dès le moment où s'est formée une intention, laquelle ne donne pas obligatoirement lieu à une action sur le système. Dans le cas où l'utilisateur effectue tout de même un début d'exécution d'une action erronée sur le système, laquelle est bloquée par une fonction de contrainte, le concepteur parlera de prévention d'erreur,

[6] Actuellement ce problème est partiellement résolu : les téléphones actuels font retentir une série de bips environ deux secondes après que le combiné ait été raccroché. Si l'utilisateur est pressé, ces deux secondes suffisent pour qu'il soit déjà parti en oubliant sa carte.

car celle-ci n'a pas atteint le noyau fonctionnel du système, et n'ouvre pas la voie à une correction de la part de l'utilisateur.

CONCLUSION

Dans cet article, nous avons défini la technique des fonctions de contrainte comme dispositif de prévention d'erreur. Nous avons montré ce que recouvre la notion de fonction de contrainte au travers d'une classification de celle-ci. Nous avons ensuite proposé un guide d'usage de cette dernière. Cependant, notre approche est essentiellement technique ne dispense en aucun cas le concepteur d'une approche ergonomique du problème.

En outre, les fonctions de contrainte ne sont qu'une des multiples réponses possibles face aux erreurs humaines. comptons rapprocher ce travail, dirigé actuellement vers une seule technique de prévention d'erreur, des notions de correction d'erreur [4, 6] et d'interruption [12], dans le but de proposer une approche unifiée des techniques de prévention, limitation, et correction d'erreur.

BIBLIOGRAPHIE

1. Amalberti, R. Facteurs humains et industrie de haute technologie à risques : ne nous trompons pas de guerre. In *Proceedings of Ergonomie et informatique avancée (ErgoIA'98)* (4-6 novembre, Biarritz, France), 1998, pp. 109-117.
2. Garmin. *GPS90 Navigateur™ Personnel*. Garmin® International, Mars 1995, Manuel d'utilisation Part#190-00084-00 Révision A.
3. Hollnagel, E., Warren, C. et Cacciabue, P.C. Automation in aerospace and aviation applications: Where are the limits. In *Proceedings of Human-Machine Interaction and Artificial Intelligence in Aerospace (HMI-AI-AS'93) Workshop* (September 28-30, Toulouse, France), 1993.
4. Jambon, F. Taxonomy for Human Error and System Fault Recovery from the Engineering Perspective. In *Proceedings of International Conference on Human-Computer Interaction in Aeronautics (HCI-Aero'98)* (May 27-29, Montréal, Canada), 1998, pp. 55-60.
5. Kolski, C. et Le Strugeon, E. A Review of Intelligent Human-Machine Interfaces in the Light of the ARCH Model. *International Journal of Human-Computer Interaction*. 3, 10 (1998), pp. 193-231.
6. Lenman, S. et Robert, J.-M. A framework for error recovery. In *Proceedings of International Ergonomics Association (IEA'94)* (August 15-19, Toronto, Canada), International Ergonomics Association, 1994, pp. 374-376.
7. Lewis, C. et Norman, D.A. Designing for Error. Norman, D. A. et Draper, S. W. (Ed.). In *User Centered System Design / New Perspectives on Human-Computer Interaction,* Lawrence Erlbaum Associates, Hillsdale NJ, USA, 1986, pp. 411-432.
8. Mo, J., Crouzet, Y., Pelud, L., Mazet, C. et Peytavin, A. Tolérance aux erreurs d'opérateur par le système technique : le prototype THEOS. In *Proceedings of 11ème Colloque National de Fiabilité & Maintenabilité* (29 Septembre - 1er Octobre, Arcachon, France), 1998, pp. 38-47.
9. Nicolet, J.-L., Carnino, A. et Wanner, J.-C. *Catastrophes ? Non merci ! La prévention des risques technologiques et humains*. Éditions Masson, Paris, France, 1990.
10. Norman, D.A. *The design of every day things*. Doubleday Currency, NewYork NY, USA, 1990.
11. Norman, D.A. Cognitive science in the cockpit. *Gateway (Crew System Information Analysis Center - University of Dayton Research Institute)*. II, 2 (Spring 1991), pp. 1-6.
12. Pérez-Quiñones, M.A. et Sibert, J.L. Negociating user-initiated cancellation and interruption requests. In *Proceedings of Conference on Human Factors in Computing Systems (CHI'96)* (14-18 April, Vancouver, Canada), ACM Press, 1996, pp. 267-268.
13. Reason, J. *Human error*. Cambridge University Press, NewYork NY, USA, 1990.
14. Reason, J. *L'erreur humaine*. Presses Universitaires de France, Paris, France, 1993.
15. Rizzo, A., Ferrante, D. et Bagnara, S. Handling Human Error. Hoc, J.-M., Cacciabue, P. C. et Hollnagel, E. (Ed.). In *Expertise and Technology: Cognition & Human-Computer Interaction*, Lawrence Erlbaum, Hillsdale NJ, USA, 1995, pp. 195-212.
16. Scapin, D. *Guide ergonomique de conception des interfaces homme-machine*. INRIA Rocquencourt, Octobre 1986, Rapport technique 77.
17. Tarnowski, E. Training Philosophy for Protected Aircraft in Emergency Situations. *FAST Airbus Technical Digest*. 23 (October 1998), pp. 2-9.
18. Zapf, D. et Reason, J.T. Introduction: Human Errors and Error Handling. *Applied Psychology: An International Review*. 43, 4 (1994), pp. 427-432.